A Software Package to Construct Polynomial Sets over $\mathbb{Z}_2$ for Determining the Output of Quantum Computations

Vladimir P. Gerdt[a*], Vasily M. Severyanov[a†]

[a]Laboratory of Information Technologies, Joint Institute for Nuclear Research,
141980 Dubna, Russia

A C# package is presented that allows a user for an input quantum circuit to generate a set of multivariate polynomials over the finite field $\mathbb{Z}_2$ whose total number of solutions in $\mathbb{Z}_2$ determines the output of the quantum computation defined by the circuit. The generated polynomial system can further be converted to the canonical Gröbner basis form which provides a universal algorithmic tool for counting the number of common roots of the polynomials.

1. INTRODUCTION

One important aspect of quantum computation is estimation of computational power of quantum logical circuits. As it was recently shown in [1], determining the output of a quantum computation is equivalent to counting the number of solutions of a certain set of polynomials defined over the finite field $\mathbb{Z}_2$.

Using ideas published in [1], we have written a C# program enabling one to assemble an arbitrary quantum circuit in a particular universal gate basis and to construct the corresponding set of polynomial equations over $\mathbb{Z}_2$. The number of solutions of the set defines the matrix elements of the circuit and therefore its output value for any input value.

The generated polynomial system can further be converted into the canonical Gröbner basis form by applying efficient involutive algorithms described in [2]. A triangular Gröbner basis for the pure lexicographical order on the polynomial variables is generally most appropriate for counting the number of common roots of the polynomials.

Our program has a user-friendly graphical interface and a built-in base of the elementary gates representing certain quantum gates and wires. A user can easily assemble an input circuit from those elements.

The structure of the paper is as follows. In Section 2 we outline shortly the circuit model of quantum computation. Section 3 presents the famous Feynman's sum-over-paths method applied to quantum circuits. In Section 4 we describe a circuit decomposition in terms of the elementary gates. In Section 5 we show how to assemble an arbitrary circuit composed from the Hadamard and Toffoli gates that form a universal basis. Section 6 demonstrates a simple example of handling the polynomials associated with a quantum circuit by constructing their Gröbner basis. We conclude in Section 7.

2. QUANTUM CIRCUITS

To quantize the classical bit, we go from the two-element set $\{0,1\}$ to a two-level quantum system described by the two-dimensional Hilbert space $\mathbb{C}^2$. In contrast to the classical case, the quantum bit (qubit) can be found in a superposition $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ of the states $|0\rangle$ and $|1\rangle$ called a computational basis, where $\alpha, \beta \in \mathbb{C}$ are the probability amplitudes of $|0\rangle$ and $|1\rangle$ respectively.

The simplest quantum computation is a unitary transformation on the qubit state

$$|\varphi\rangle = U\,|\psi\rangle\,, \qquad UU^\dagger = I.$$

A measurement of the qubit in the computational basis $|0\rangle$ and $|1\rangle$ transforms its state to one of the

[*] gerdt@jinr.ru

[†] severyan@jinr.ru

basis states with probabilities determined by the amplitudes

$$\alpha|0\rangle + \beta|1\rangle \mapsto \begin{cases} |0\rangle \text{ with probability } |\alpha|^2 \\ |1\rangle \text{ with probability } |\beta|^2 \end{cases}$$

To compute a reversible Boolean vector-function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$, one applies the appropriate unitary transformation U_f to an input state $|\mathbf{a}\rangle$ composed of some number of qubits

$$|\mathbf{b}\rangle = U_f\,|\mathbf{a}\rangle\,, \qquad |\mathbf{a}\rangle\,, |\mathbf{b}\rangle \in \mathbb{C}^{2\otimes n}$$

The output state $|\mathbf{b}\rangle$ is not the outcome of the computation until its measurement. After that the output state can be used anywhere.

Some unitary transformations are called quantum gates. A quantum gate acts only on a few qubits, on the rest it acts as the identity. A quantum circuit can be assembled by appropriately aligning quantum gates. The unitary transformation defined by the circuit is the composition of the constituent unitary transformations

$$U_f = U_m U_{m-1} \cdots U_2 U_1 \tag{1}$$

A quantum gate basis is a set of universal quantum gates, i.e. any unitary transformation can be presented as a composition of the gates of the basis. As well as in the classical case, there are several sets of universal quantum gates. For our work it is convenient to choose the particular universal gate basis consisting of Hadamard and Toffoli gates [3].

The Hadamard gate is a one-qubit gate. It turns a computational basis state into the equally weighted superposition

$$H : |0\rangle \mapsto \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$H : |1\rangle \mapsto \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

The resulting superpositions for $|0\rangle$ and $|1\rangle$ differ by a phase factor.

The Toffoli gate is a tree-qubit gate. Input bits x and y control the behavior of bit z, and the Toffoli gate acts on computational basis states as

$$(x, y, z) \mapsto (x, y, z \oplus xy)$$

An action of a quantum circuit can be described by a square unitary matrix whose matrix element $\langle \mathbf{b}|\, U_f\, |\mathbf{a}\rangle$ yields the probability amplitude for transition from an initial quantum state $|\mathbf{a}\rangle$ to the final quantum state $|\mathbf{b}\rangle$. The matrix element is decomposed in accordance to the gate decomposition of the circuit unitary transformation (1) and can be calculated as sum over all the intermediate states $\mathbf{a}_i$, i = 1,2, . . . m - 1:

$$\langle \mathbf{b}|\, U_f\, |\mathbf{a}\rangle = \sum_{\mathbf{a}_i} \langle \mathbf{b}|\, U_m\, |\mathbf{a}_{m-1}\rangle \cdots \langle \mathbf{a}_1|\, U_1\, |\mathbf{a}\rangle$$

3. FEYNMAN'S SUM-OVER-PATHS

To apply the famous Feynman's sum-over-paths approach to calculate the matrix element of a quantum circuit, we replace every quantum gate of the circuit under consideration by its classical counterpart. The trick here is to select the corresponding classical gate for the quantum Hadamard gate because for any input value, 0 or 1, it gives with equal probability either 0 or 1. We denote the output of the classical Hadamard gate by the path variable x. Its value determines one of the two possible paths of computation. The classical Toffoli gate acts as

$$(a_1, a_2, a_3) \mapsto (a_1, a_2, a_3 \oplus a_1 a_2)\,,$$

and the classical Hadamard gate as

$$a_1 \mapsto x \qquad a_i, x \in \mathbb{Z}_2$$

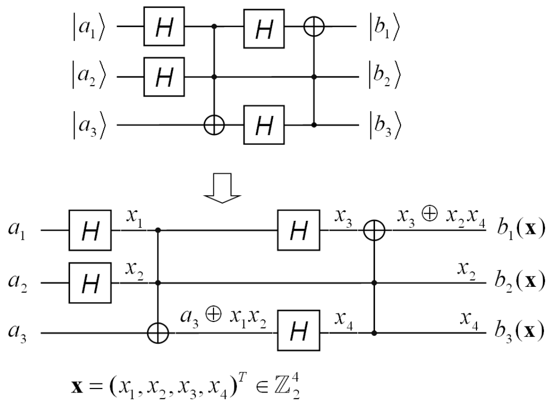


Figure 1. From quantum to classical circuit

Fig. 1 shows an example of quantum circuit (taken from [1]) and its classical correspondence. The path variables x_i comprise the (vector) path $\mathbf{x} = (x_1, x_2, x_3, x_4)^T \in \mathbb{Z}_2^4$.

A classical path is a sequence of classical bit strings $a, a_1, a_2, \ldots, a_m = b$ resulting from application of the classical gates. For each selection of values for the path variables x_i we have a sequence of classical bit strings which is called an admissible classical path. Each admissible classical path has a phase which is determined by the Hadamard gates applied. The phase is changed only when the input and output of the Hadamard gate are simultaneously equal to 1, and this gives the folmula

$$\varphi(\mathbf{x}) = \sum_{\text{Hadamardgates}} \mathit{input} \bullet \mathit{output}$$

Toffoli gates do not change the phase.

For our example the phase of the path $\mathbf{x}$ is

$$\varphi(\mathbf{x}) = a_1x_1 \oplus a_2x_2 \oplus x_1x_3 \oplus x_4(a_3 \oplus x_1x_2)$$

The matrix element of a quantum circuit is given by sum over all the allowed paths from the classical states $\mathbf{a}$ to $\mathbf{b}$

$$\langle \mathbf{b} | \, U_f \, | \mathbf{a} \rangle = \frac{1}{\sqrt{2^h}} \sum_{\mathbf{x}:\mathbf{b}(\mathbf{x})=\mathbf{b}} (-1)^{\varphi(\mathbf{x})}$$

where h is the number of Hadamard gates. The terms in the sum have the same absolute value but vary in sign.

Let N_0 be the number of positive terms in the sum and N_1 the number of negative terms

$$N_0 = |\{x | b(x) = b \quad \& \quad \varphi(x) = 0\}|$$

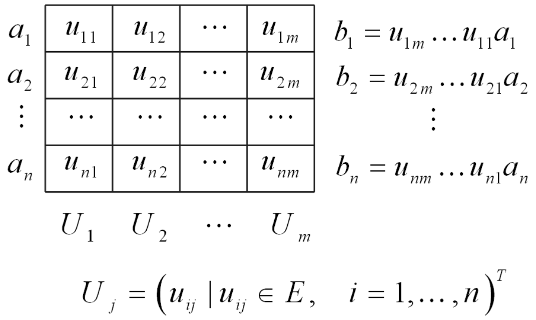
$$N_1 = |\{x | b(x) = b \quad \& \quad \varphi(x) = 1\}|$$

These equations count solutions to a system of $n+1$ polynomials in h variables over $\mathbb{Z}_2$. Then the matrix element may be written as the difference

$$\langle \mathbf{b} | \, U_f \, | \mathbf{a} \rangle = \frac{1}{\sqrt{2^h}} (N_0 - N_1)$$

4. CIRCUIT DECOMPOSITION

To provide a user with a tool for assembling arbitrary quantum circuits composed from the Hadamard and Toffoli gates we represent a circuit as a rectangular table (Fig. 2).



$$\begin{array}{c|c|c|c|c|l}
\cline{2-5}
a_1 & u_{11} & u_{12} & \cdots & u_{1m} & b_1 = u_{1m} \ldots u_{11} a_1 \\
\cline{2-5}
a_2 & u_{21} & u_{22} & \cdots & u_{2m} & b_2 = u_{2m} \ldots u_{21} a_2 \\
\cline{2-5}
\vdots & \cdots & \cdots & \cdots & \cdots & \qquad \vdots \\
\cline{2-5}
a_n & u_{n1} & u_{n2} & \cdots & u_{nm} & b_n = u_{nm} \ldots u_{n1} a_n \\
\cline{2-5}
\multicolumn{1}{c}{} & \multicolumn{1}{c}{U_1} & \multicolumn{1}{c}{U_2} & \multicolumn{1}{c}{\cdots} & \multicolumn{1}{c}{U_m} &
\end{array}$$

$$U_j = \left(u_{ij} \mid u_{ij} \in E, \quad i = 1, \ldots, n \right)^T$$

Figure 2. Circuit decomposition into elementary gates

Each cell in the table contains an elementary gate from following set

$$E = \{I, \overset{+}{I}, \hat{I}, \overset{\vee}{I}, \hat{M}, \overset{\vee}{M}, \hat{A}, \overset{\vee}{A}, H\} \tag{2}$$

so that the output for each row is determined by the composition of the elementary gates in the row. Thereby, each elementary unitary transformation U_j is represented as an n-tuple of elementary gates.

Fig. 3 shows action of the elementary gates from (2): the identities, the multiplications, the additions modulo 2, and the classical Hadamard gate. The identity just reproduces its input. The identity-cross reproduces also its vertical input from the top elementary gate to the bottom one and vice versa. Every identity-down and identity-up have two outputs – horizontal and vertical. The multiplication-up and multiplication-down perform multiplication of their horizontal and the corresponding vertical inputs. In a similar manner act the addition-up and addition-down. Each Hadamard gate outputs an independent path variable irrespective of its input and can give a nonzero contribution to the phase.

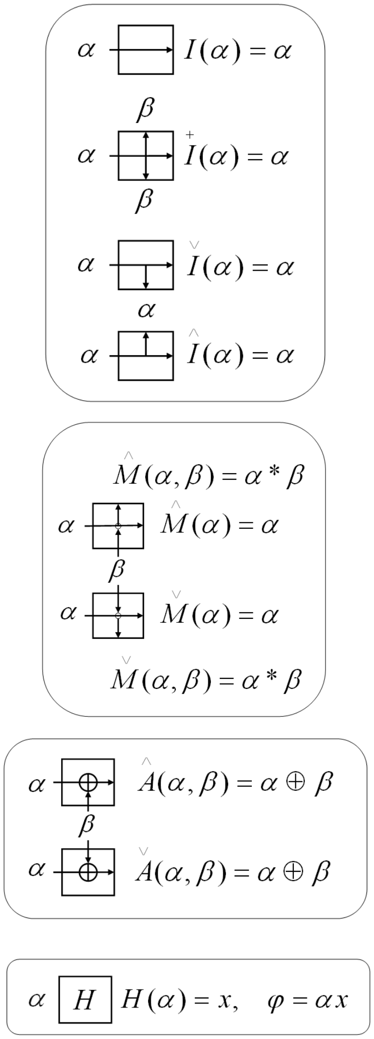

α
$I(\alpha)=\alpha$
β
$\overset{+}{I}(\alpha)=\alpha$
$\check{I}(\alpha)=\alpha$
$\hat{I}(\alpha)=\alpha$
$\hat{M}(\alpha,\beta)=\alpha*\beta$
$\hat{M}(\alpha)=\alpha$
$\check{M}(\alpha)=\alpha$
$\check{M}(\alpha,\beta)=\alpha*\beta$
$\hat{A}(\alpha,\beta)=\alpha\oplus\beta$
$\check{A}(\alpha,\beta)=\alpha\oplus\beta$
H
$H(\alpha)=x,\quad \varphi=\alpha x$

Figure 3. Action of elementary gates

5. ASSEMBLING CIRCUITS

How can one assemble a circuit? First of all, we define an empty table of the required size. In this case both output and phase are not fixed. Then we place the required elementary gates in appropriate cells. Now the output is the result of applying the elementary gates to the input. The phase is also calculated. Then we proceed the same way with the second column, with the third column, and so on up to the last column. Fig. 4 shows an example.

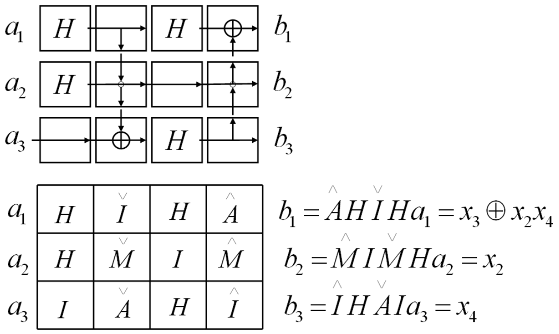

a_1
a_2
a_3
H
b_1
b_2
b_3
H $\check{I}$ H $\hat{A}$
H $\check{M}$ I $\hat{M}$
I $\check{A}$ H $\hat{I}$
$b_1=\hat{A}H\check{I}Ha_1=x_3\oplus x_2x_4$
$b_2=\hat{M}I\check{M}Ha_2=x_2$
$b_3=\hat{I}H\check{A}Ia_3=x_4$

Figure 4. Assembling circuit

Apart from the ordinary menu, our program contains the toolbar for selecting an elementary gate and the toolbar for main operations. There are two windows: for assembling a circuit and for showing its polynomials.

A circuit is represented in the program as two $2d$-arrays: one for the elementary gates and another for their polynomials. The phase polynomial is separately represented. The following piece of code demonstrates construction of the circuit polynomials

```
for each Column in Table of Gates
  for each Gate in Column {
    construct Gate Polynomial;
    if Gate id Hadamard
      reconstruct Phase Polynomial; }
```

The method for constructing a gate polynomial is recursive because of the need to go up or down for some gates.

Any circuit is saved as two files. One file is binary and contains the circuit itself. Another file has a text format. It contains the circuit polynomials in a symbolic form. The program allows to save polynomials in several formats convenient for loading into a computer algebra system (for example, in Maple or Mathematica) for the further processing. It is also possible to load back in memory a saved circuit.

Note, that the part of our code for name space **Polynomial_Modulo_2** written in C# [4] can also be used independently on our program. This part contains classes for handling polynomials over the finite field $\mathbb{Z}_2$. Class Polynomial is a list of monomials, class Monomial is a list of letters, class Letter is an indexed letter provided with a positive integer superscript (power degree).

6. QUANTUM POLYNOMIALS

A system generated by the program is a finite set $F \subset R$ of polynomials in the ring

$$\begin{array}{l} R := \mathbb{Z}_2[a_i, b_j][x_1, ..., x_h] \\ a_i, b_j \in \ \mathbb{Z}_2, \ \ i, j = 1, ...n \end{array}$$

in h variables and $2n$ binary coefficients. One has to count the number of roots N_0 and N_1 in $\mathbb{Z}_2$ of the polynomial sets

$$F_0 = \{f, ..., f_k, \varphi\} \ , \ \ F_1 = \{f, ..., f_k, \varphi + 1\}$$

Then the circuit matrix is given by

$$\langle \mathbf{b} | \ U \ | \mathbf{a} \rangle = \frac{1}{\sqrt{2^h}} \ (N_0 - N_1)$$

To count the number of roots one can convert F_0 and F_1 into a triangular form by computing the lexicographical Gröbner basis by means of the Buchberger algorithm or by involutive algorithm decribed in [2].

For the example shown on Fig. 1 we have the following polynomial system:

$$\begin{array}{l} f_1 = x_2x_4 \ + x_3 + b_1 \\ f_2 = x_2 + b_2 \\ f_3 = x_4 + b_3 \\ \varphi = \ x_1x_2 + x_1x_3 + a_1x_1 + a_2x_2 + a_3x_4 \end{array}$$

The lexicographical Gröbner basis for the ordering $x_1 \succ \ x_2 \succ \ x_3 \succ \ x_4$ on the variables and representing both F_0 and F_1 is as follows

$$\begin{array}{l} g_1 = (a_1 + b_1)x_1 \ + a_2b_2 + a_3b_3 \ (+1) \\ g_2 = x_2 + b_2 \\ g_3 = \ x_3 + b_1 + b_2b_3 \\ g_3 = x_4 + b_3 \end{array}$$

From this lexicographical Gröbner basis we immediately obtain the following conditions on the parameters:

$$\begin{array}{l} a_1 + b_1 = 0 \ \ \& \ \ a_2b_2 + a_3b_3 = 0 \\ a_1 + b_1 = 0 \ \ \& \ \ a_2b_2 + a_3b_3 = 1 \end{array}$$

From these conditions we easily count 2 (0) roots of F_0 (F_1) and 0 (2) roots of F_0 (F_1). In all other cases there is 1 root of F_0 and F_1.

Some matrix elements are

$$\langle 000 | \, U \, | 001 \rangle = \frac{1}{2} \ , \ \ \langle 000 | \, U \, | 111 \rangle = 0$$

7. CONCLUSION

We presented the first version of a program tool for assembling arbitrary quantum circuits and for constructing the corresponding polynomial equation systems. Its number of solutions uniquely determines the circuit matrix.

There is the algorithmic Gröbner basis approach to converting the system of quantum polynomials into a triangular form which is useful for computing the number of solutions.

Thus, the above presented software together with Gröbner bases provide a tool for simulating quantum circuits.

8. ACKNOWLEDGMENTS

The research presented in this paper was partially supported by the grant 04-01-00784 from the Russian Foundation for Basic Research.

REFERENCES

1. Christopher M. Dawson et al. *Quantum computing and polynomial equations over the finite field $\mathbb{Z}_2$*. arXiv:quant-ph/0408129.
2. Gerdt V.P. *Involutive Algorithms for Computing Grobner Bases*. Computational commutative and non-commutative algebraic geometry, IOS Press, Amsterdam, 2005, pp.199-225. arXiv:math.AC/0501111, 2005.
3. Aharonov D. *A Simple Proof that Toffoli and Hadamard Gatesare Quantum Universal.* arXiv:quant-ph/0301040.
4. *Microsoft Visual C# .net Standard*, Version 2003.